भारतीय प्रौद्योगिकी संस्थान जम्मू

INDIAN INSTITUTE OF TECHNOLOGY JAMMU

Major Project

Radix-4 Modified Booth Multiplier

Submitted by: Irfan Ahmad(2023PVL0089) & Md Mahtab Alam(2023PVL0093)

Submitted to: Assistant Professor Satyadev Ahlawat

November 2023

Introduction

A Radix-4 Booth Multiplier is a specific implementation of the Booth algorithm designed to further optimize the multiplication process by utilizing a radix-4 numeral system. It's an extension of the Radix-2 Booth Multiplier, which is based on a binary numeral system.

Problem Statement

Design a high-speed Radix-4 Multiplier with the following specifications:

1. The multiplier shall be based on Modified Booth Encoding Algorithm.

2. The operand size must be equal to 2 Bytes each.

3. Design the Multiplier in such a way that it can handle both 2s complement and

unsigned multipliers. For addition of the Partial Products you have to use Carry

Save Adder based implementation.

4. Add adequate comments in your VHDL code for readability.

Approach

A Radix-4 Booth Multiplier is a hardware circuit that performs multiplication using a radix-4 Booth encoding scheme. This encoding scheme reduces the number of partial products that need to be generated compared to a regular multiplier. The basic idea is to use a set of precomputed partial products and selectively add or subtract them based on the Booth encoding of the multiplier.

.

VHDL code of the simulation of the Radix 4-Booth Multiplier.

1) *Code for generation of the booth encoded partial products*.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity BoothMultiplier is
  Port (
    multiplier : in STD_LOGIC_VECTOR(15 downto 0);
    multplcnt : in STD_LOGIC_VECTOR(15 downto 0);
    Partial_product_1 : buffer STD_LOGIC_VECTOR(31 downto 0);
    Partial_product_2 : buffer STD_LOGIC_VECTOR(31 downto 0);
    Partial_product_3 : buffer STD_LOGIC_VECTOR(31 downto 0);
    Partial_product_4 : buffer STD_LOGIC_VECTOR(31 downto 0);
    Partial_product_5 : buffer STD_LOGIC_VECTOR(31 downto 0);
    Partial_product_6 : buffer STD_LOGIC_VECTOR(31 downto 0);
    Partial_product_7 : buffer STD_LOGIC_VECTOR(31 downto 0);
    Partial_product_8 : buffer STD_LOGIC_VECTOR(31 downto 0)


  );
end BoothMultiplier;

architecture Behavioral of BoothMultiplier is

begin


  process(multiplier)
  begin
   -- Initialization of partial_product----
    Partial_product_1 <= (others => '0');
      case multiplier(1 downto 0) is
        when "00" =>
          Partial_product_1 <= (others => '0');
        when "01" =>
          Partial_product_1(31 downto 0) <= Partial_product_1(31 downto 16)& 
multplcnt;
        when "10" =>
         Partial_product_1(31 downto 0) <=not Partial_product_1(31 downto 17) & ( not 
multplcnt +1)&'0';
        when "11" =>
```

```
          Partial_product_1(31 downto 0) <= not Partial_product_1(31 downto 16)&(not
multplcnt +1);
       when others =>


        Partial_product_1 <= (others => '0');
      end case;

  end process;

  -- Partial_product_2
   process(multiplier)
  begin

    Partial_product_2 <= (others => '0');

      case multiplier(3 downto 1) is
        when "000"|"111" =>
          Partial_product_2 <= (others => '0');
        when "001" |"010" =>
          Partial_product_2(31 downto 0) <=Partial_product_2 (31 downto 18)&
multplcnt&"00";
        when "011" =>
          Partial_product_2(31 downto 0 ) <= Partial_product_2 (31 downto 19)&
multplcnt & "000";
        when "100" =>
         Partial_product_2(31 downto 0) <=not Partial_product_2 (31 downto 19)&( not
multplcnt +1)&"000";


        when "101"| "110" =>
          Partial_product_2(31 downto 0) <= not Partial_product_2 (31 downto 18)&not
multplcnt +1&"00";
        when others =>
        Partial_product_2 <= (others => '0');
      end case;

  end process;
  -- Partial_product_3
   process(multiplier)
  begin

    Partial_product_3 <= (others => '0');

      case multiplier(5 downto 3) is
        when "000"|"111" =>
          Partial_product_3 <= (others => '0');
        when "001" |"010" =>
          Partial_product_3(31   downto   0)   <=Partial_product_3   (31   downto   20)&
multplcnt&"0000";
        when "011" =>
          Partial_product_3(31   downto   0   )   <=   Partial_product_3   (31   downto   21)&
multplcnt & "00000";
        when "100" =>
         Partial_product_3(31 downto 0) <=not Partial_product_3 (31 downto 21)&( not
multplcnt +1)&"00000";


        when "101"| "110" =>
          Partial_product_3(31 downto 0) <= not Partial_product_3 (31 downto 20)&not
multplcnt +1&"0000";
        when others =>

          Partial_product_3 <= (others => '0');
      end case;

  end process;
  -- Partial_product_4
   process(multiplier)
  begin

    Partial_product_4 <= (others => '0');

      case multiplier(7 downto 5) is
        when "000"|"111" =>
          Partial_product_4 <= (others => '0');
        when "001" |"010" =>
          Partial_product_4(31 downto 0) <=Partial_product_4 (31 downto 22)&
multplcnt &"000000";
        when "011" =>
          Partial_product_4(31 downto 0 ) <=Partial_product_4 (31 downto 23)&
multplcnt & "0000000";
        when "100" =>
         Partial_product_4(31 downto 0) <=not Partial_product_4 (31 downto 23)&( not
multplcnt +1)&"0000000";


        when "101"| "110" =>
          Partial_product_4(31 downto 0) <= not Partial_product_4 (31 downto 22)&not
multplcnt +1&"000000";
        when others =>

          Partial_product_4 <= (others => '0');
      end case;

  end process;
   -- Partial_product_5
   process(multiplier)
  begin

    Partial_product_5 <= (others => '0');
```

```
    case multiplier(9 downto 7) is
      when "000"|"111" =>
        Partial_product_5 <= (others => '0');
      when "001" |"010" =>
        Partial_product_5(31 downto 0) <=Partial_product_5 (31 downto 24)&
multplcnt&"00000000";
      when "011" =>
        Partial_product_5(31 downto 0 ) <= Partial_product_5 (31 downto 25)&
multplcnt & "000000000";
      when "100" =>
       Partial_product_5(31 downto 0) <=not Partial_product_5 (31 downto 25)&( not
multplcnt +1)&"000000000";
      when "101"| "110" =>
        Partial_product_5(31 downto 0) <= not Partial_product_5 (31 downto 24)&not
multplcnt +1&"00000000";
      when others =>
        Partial_product_5 <= (others => '0');
    end case;

  end process;
  -- Partial_product_6
   process(multiplier)
  begin
    Partial_product_6 <= (others => '0');
    case multiplier(11 downto 9) is
      when "000"|"111" =>
        Partial_product_6 <= (others => '0');
      when "001" |"010" =>
        Partial_product_6(31 downto 0) <=Partial_product_6 (31 downto 26)&
multplcnt&"0000000000";
      when "011" =>
        Partial_product_6(31 downto 0 ) <= Partial_product_6 (31 downto 27)&
multplcnt & "00000000000";
      when "100" =>
```

```
         Partial_product_6(31 downto 0) <=not Partial_product_6 (31 downto 27)&( not
multplcnt +1)&"00000000000";


        when "101"| "110" =>
          Partial_product_6(31 downto 0) <= not Partial_product_6 (31 downto 26)&not
multplcnt +1&"0000000000";
        when others =>

          Partial_product_6 <= (others => '0');
      end case;

  end process;
  -- Partial_product_7
   process(multiplier)
  begin

    Partial_product_7 <= (others => '0');


      case multiplier(13 downto 11) is
        when "000"|"111" =>
          Partial_product_7 <= (others => '0');
        when "001" |"010" =>
          Partial_product_7(31 downto 0) <=Partial_product_7 (31 downto 28)&
multplcnt &"000000000000";
        when "011" =>
          Partial_product_7(31 downto 0 ) <=Partial_product_7 (31 downto 29)&
multplcnt & "0000000000000";
        when "100" =>
         Partial_product_7(31 downto 0) <=not Partial_product_7 (31 downto 29)&( not
multplcnt +1)&"0000000000000";
        when "101"| "110" =>
        Partial_product_7(31 downto 0) <= not Partial_product_7 (31 downto 28)&not
multplcnt +1&"000000000000";
       when others =>
        Partial_product_7 <= (others => '0');
     end case;

  -- Partial_product_8
  end process;

process(multiplier)
  begin

    Partial_product_8 <= (others => '0');

     case multiplier(15 downto 13) is
       when "000"|"111" =>
        Partial_product_8 <= (others => '0');
       when "001" |"010" =>
        Partial_product_8(31   downto   0)   <=Partial_product_8   (31   downto   30)&
multplcnt&"00000000000000";
       when "011" =>
        Partial_product_8(31  downto  0  )  <=  Partial_product_8  (31)&   multplcnt  &
"000000000000000";
       when "100" =>
       Partial_product_8(31 downto 0) <=not Partial_product_8 (31)&( not multplcnt
+1)&"000000000000000";

       when "101"| "110" =>
        Partial_product_8(31 downto 0) <= not Partial_product_8 (31 downto 30)&not
multplcnt +1&"00000000000000";
       when others =>

        Partial_product_8 <= (others => '0');
     end case;
    end process;

   end Behavioral;
```

2) *Code of the addition of the partial products using Carry Save adder with a last stage of ripple carry adder.*

```
-------------------Carry Save adder and ripple carry adder section to sum partial products

library ieee;

use ieee.std_logic_1164.all;

use ieee.numeric_std.all;


entity booth_multiplier is

port(partial_product_1,partial_product_2,partial_product_3,partial_product_4,partial_product
_5,partial_product_6,partial_product_7,partial_product_8 : in std_logic_vector(31 downto 0));

 end booth_multiplier;


architecture arch of booth_multiplier is

signal co_csa1 : std_logic_vector(31 downto 0);

signal co_csa2 : std_logic_vector(31 downto 0);

signal co_csa3 : std_logic_vector(31 downto 0);

signal co_csa4 : std_logic_vector(31 downto 0);

signal co_csa5 : std_logic_vector(31 downto 0);

signal co_csa6 : std_logic_vector(31 downto 0);

signal sum_csa1 : std_logic_vector(31 downto 0);

signal sum_csa2 : std_logic_vector(31 downto 0);

signal sum_csa3 : std_logic_vector(31 downto 0);

signal sum_csa4 : std_logic_vector(31 downto 0);
signal sum_csa5 : std_logic_vector(31 downto 0);

signal sum_csa6 : std_logic_vector(31 downto 0);


SIGNAL rca_sum : STD_LOGIC_VECTOR(31 DOWNTO 0);

SIGNAL rca_cout : STD_LOGIC;

signal cin_to_rca:  std_logic_vector(31 downto 0);


component CSA is

  port (p,q,r:in std_logic_vector(31 downto 0);

  sm,cr: buffer std_logic_vector(31 downto 0));

end component;

COMPONENT rca is

        PORT (

            a, b: IN STD_LOGIC_VECTOR(31 DOWNTO 0);

            cin: IN STD_LOGIC;

            sout: OUT STD_LOGIC_VECTOR(31 DOWNTO 0);

            cout: OUT STD_LOGIC

        );

    END COMPONENT;


  begin

  --stage1

    a0:  CSA  port  map  (p  =>  partial_product_1,q  =>  partial_product_2,r  =>
partial_product_3,sm => sum_csa1,cr => co_csa1);

   --stage2
```

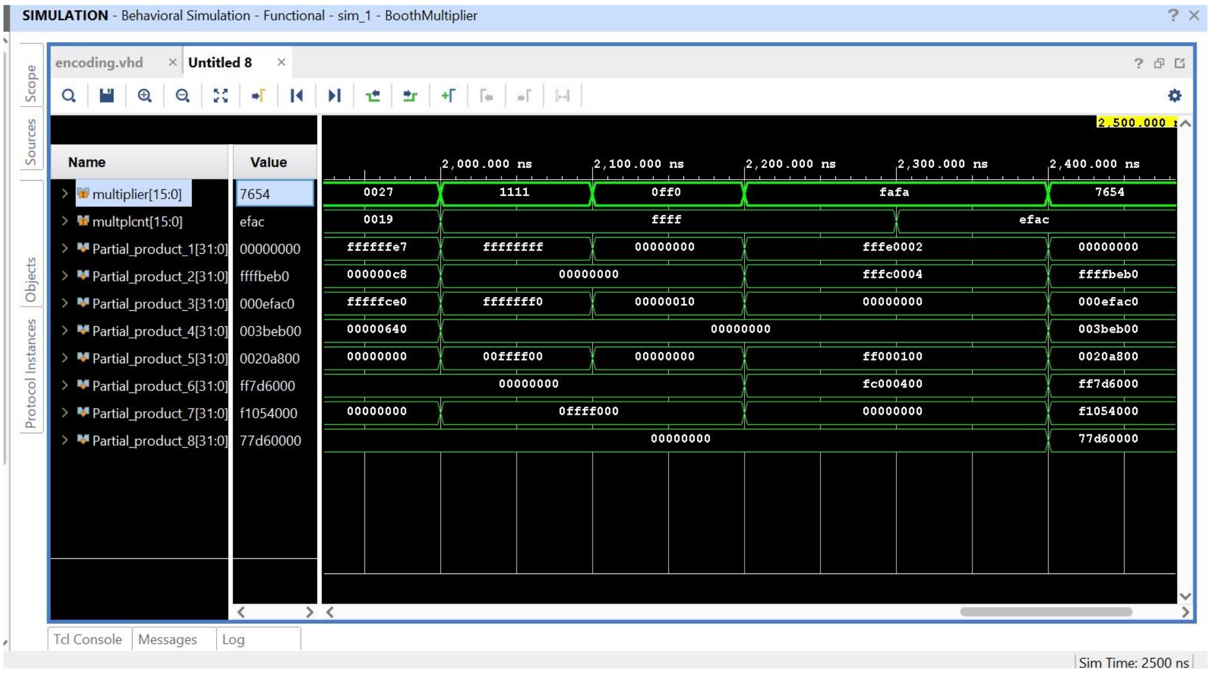
```
    a1:CSA port map (p => sum_csa1,q => co_csa1,r => partial_product_4,sm => sum_csa2,cr
=> co_csa2);
  --stage3
    a3:CSA port map (p => sum_csa2 , q => co_csa2,r => partial_product_5, sm => sum_csa3
,cr => co_csa3);
    --stage 4
    a4:CSA port map (p => sum_csa3,q => co_csa3,r => partial_product_6,sm => sum_csa4,cr
=> co_csa4);
    --stage5
    a5: CSA port map (p => sum_csa4,q => co_csa4,r => partial_product_7,sm => sum_csa5,cr
=> co_csa5);
    --stage6
    a6: CSA port map (p => sum_csa5 ,q => co_csa5,r => partial_product_8,sm => sum_csa6,cr
=> co_csa6);
  ------32 16 bit ripple carry adder
    Ripple_carry_adder : rca PORT MAP (a => sum_csa6, b => co_csa6, cin => '0', sout
=>rca_sum, cout => rca_cout);
  end arch;


library ieee;
use ieee.std_logic_1164.all;
entity CSA is
  port (p,q,r : in std_logic_vector(31 downto 0);
  sm,cr: buffer std_logic_vector(31 downto 0));
end CSA;
```

```
architecture behavioural of CSA is
 begin
 PROCESS(p, q, r)
 VARIABLE c: STD_LOGIC_VECTOR(31 DOWNTO 0);
 BEGIN

 FOR i IN 0 TO 31 LOOP
 sm(i) <= p(i) XOR q(i) XOR r(i);
 c(i) := (p(i) AND q(i)) OR (p(i) AND r(i)) OR (q(i) AND r(i));
 END LOOP;
 cr <= c;

 end process;
 end behavioural;

 ------ripple carry adder
LIBRARY ieee;
USE ieee.std_logic_1164.all;
 entity rca IS

 port (a, b: IN STD_LOGIC_VECTOR(31 DOWNTO 0);
 cin: IN STD_LOGIC;
 sout: OUT STD_LOGIC_VECTOR(31 DOWNTO 0);
 cout: OUT STD_LOGIC);
```

```
end rca;

 --------------------------------------------------------

ARCHITECTURE structure OF rca IS

signal cin_to_rca:  std_logic_vector(31 downto 0);


BEGIN

 PROCESS(a, b, cin)

 variable d: STD_LOGIC_VECTOR(31 DOWNTO 0);

 BEGIN

 d(0) := cin ;

 FOR i IN 1 TO 31 LOOP

 sout(i) <= a(i) XOR b(i) XOR d(i);

 d(i) := (a(i) AND b(i)) OR (a(i) AND d(i-1)) OR (b(i) AND d(i-1));

 END LOOP;

 cout <= d(31);

 cin_to_rca <= d;

 END PROCESS;

 END ARCHITECTURE;
```

Results

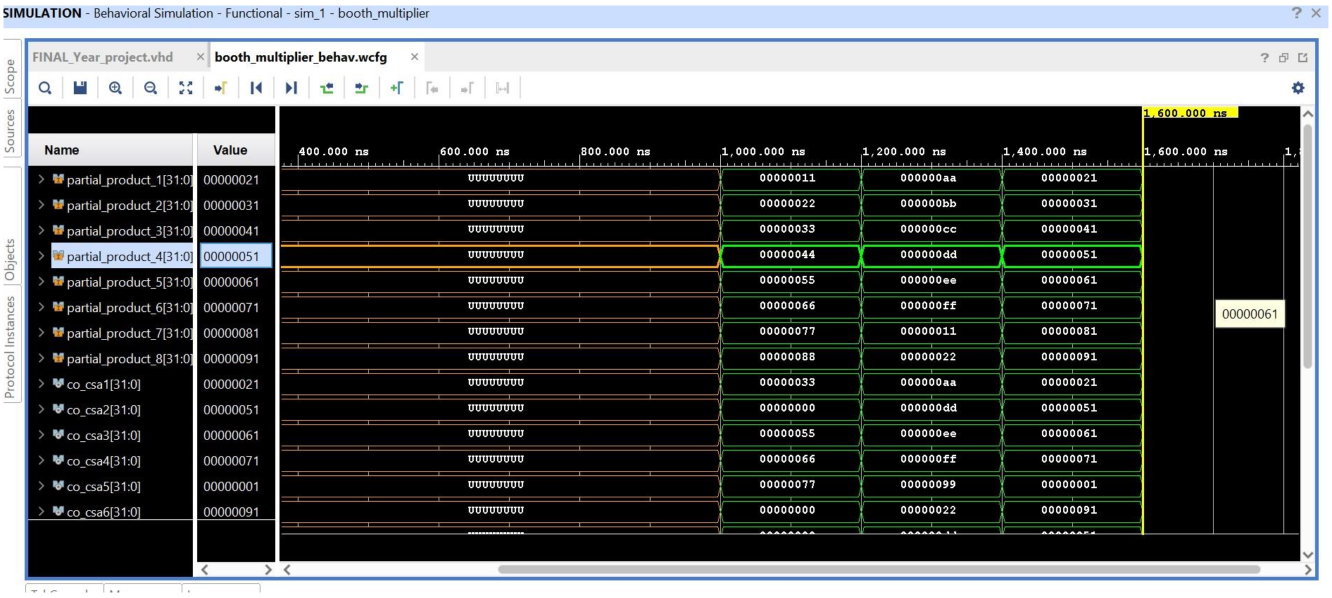
1) Generation of Booth Encoded partial products.



```
SIMULATION - Behavioral Simulation - Functional - sim_1 - BoothMultiplier

Name                     Value      2,000.000 ns   2,100.000 ns   2,200.000 ns   2,300.000 ns   2,400.000 ns
multiplier[15:0]         7654       0027 | 1111 | 0ff0 | fafa | 7654
multplcnt[15:0]          efac       0019 | ffff | efac
Partial_product_1[31:0]  00000000   ffffffe7 | ffffffff | 00000000 | fffe0002 | 00000000
Partial_product_2[31:0]  ffffbeb0   000000c8 | 00000000 | fffc0004 | ffffbeb0
Partial_product_3[31:0]  000efac0   fffffce0 | fffffff0 | 00000010 | 00000000 | 000efac0
Partial_product_4[31:0]  003beb00   00000640 | 00000000 | 003beb00
Partial_product_5[31:0]  0020a800   00000000 | 00ffff00 | 00000000 | ff000100 | 0020a800
Partial_product_6[31:0]  ff7d6000   00000000 | fc000400 | ff7d6000
Partial_product_7[31:0]  f1054000   00000000 | 0ffff000 | 00000000 | f1054000
Partial_product_8[31:0]  77d60000   00000000 | 77d60000

Sim Time: 2500 ns
```

Figure2: Generation of Booth Encoded partial products.

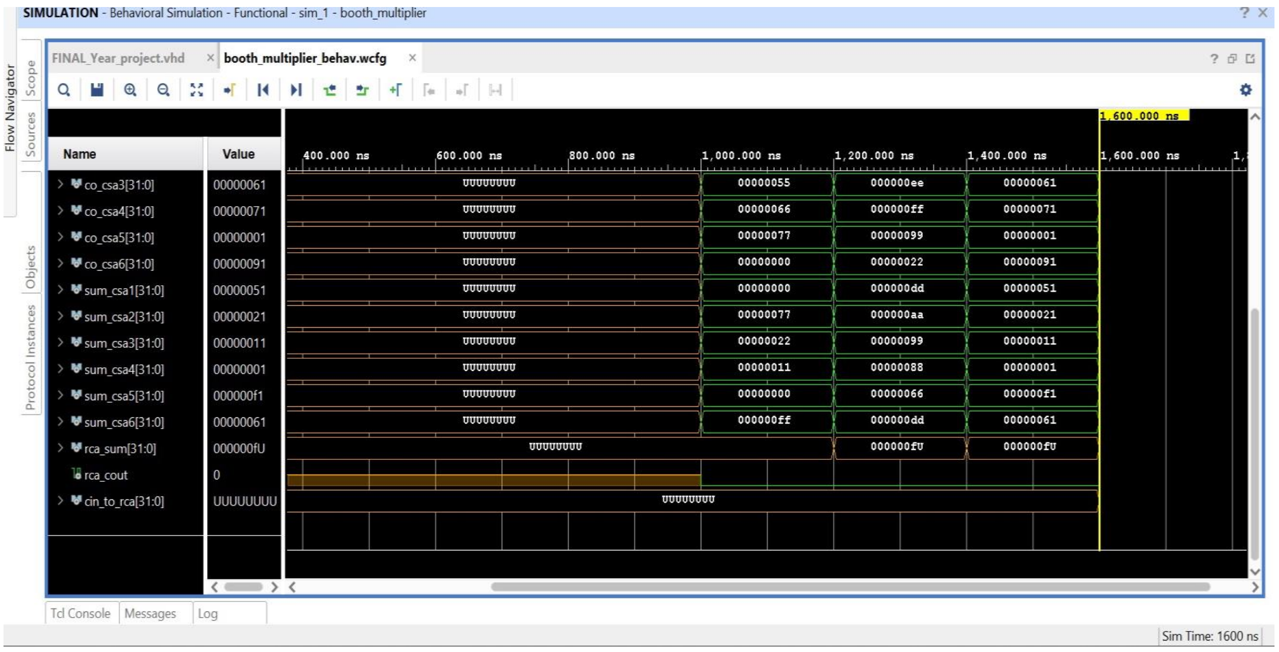
2) Addition of the partial products using Carry Save Adders.



```
SIMULATION - Behavioral Simulation - Functional - sim_1 - booth_multiplier

Name                     Value      400.000 ns  600.000 ns  800.000 ns  1,000.000 ns  1,200.000 ns  1,400.000 ns  1,600.000 ns
partial_product_1[31:0]  00000021   UUUUUUUU | 00000011 | 000000aa | 00000021
partial_product_2[31:0]  00000031   UUUUUUUU | 00000022 | 000000bb | 00000031
partial_product_3[31:0]  00000041   UUUUUUUU | 00000033 | 000000cc | 00000041
partial_product_4[31:0]  00000051   UUUUUUUU | 00000044 | 000000dd | 00000051
partial_product_5[31:0]  00000061   UUUUUUUU | 00000055 | 000000ee | 00000061
partial_product_6[31:0]  00000071   UUUUUUUU | 00000066 | 000000ff | 00000071
partial_product_7[31:0]  00000081   UUUUUUUU | 00000077 | 00000011 | 00000081
partial_product_8[31:0]  00000091   UUUUUUUU | 00000088 | 00000022 | 00000091
co_csa1[31:0]            00000021   UUUUUUUU | 00000033 | 000000aa | 00000021
co_csa2[31:0]            00000051   UUUUUUUU | 00000000 | 000000dd | 00000051
co_csa3[31:0]            00000061   UUUUUUUU | 00000055 | 000000ee | 00000061
co_csa4[31:0]            00000071   UUUUUUUU | 00000066 | 000000ff | 00000071
co_csa5[31:0]            00000001   UUUUUUUU | 00000077 | 00000099 | 00000001
co_csa6[31:0]            00000091   UUUUUUUU | 00000000 | 00000022 | 00000091

00000061
```

Figure 3: Inputs to the Carry Save Adder which are the Booth Encoded partial products.

3) Output of the Carry Save Adder.



Name	Value	400.000 ns – 1,000.000 ns	1,000.000 ns	1,200.000 ns	1,400.000 ns
co_csa3[31:0]	00000061	UUUUUUUU	00000055	000000ee	00000061
co_csa4[31:0]	00000071	UUUUUUUU	00000066	000000ff	00000071
co_csa5[31:0]	00000001	UUUUUUUU	00000077	00000099	00000001
co_csa6[31:0]	00000091	UUUUUUUU	00000000	00000022	00000091
sum_csa1[31:0]	00000051	UUUUUUUU	00000000	000000dd	00000051
sum_csa2[31:0]	00000021	UUUUUUUU	00000077	000000aa	00000021
sum_csa3[31:0]	00000011	UUUUUUUU	00000022	00000099	00000011
sum_csa4[31:0]	00000001	UUUUUUUU	00000011	00000088	00000001
sum_csa5[31:0]	000000f1	UUUUUUUU	00000000	00000066	000000f1
sum_csa6[31:0]	00000061	UUUUUUUU	000000ff	000000dd	00000061
rca_sum[31:0]	000000fU	UUUUUUUU	UUUUUUUU	000000fU	000000fU
rca_cout	0				
cin_to_rca[31:0]	UUUUUUUU	UUUUUUUU	UUUUUUUU	UUUUUUUU	UUUUUUUU

Figure 4: Intermediate signals of the Carry Save Adder producing inputs to the Ripple Carry Adder

Conclusion

Multiplication is done very faster as the number of the partial products are reduced.